VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017

Lukáš Machala

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Machala**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Craneballs s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Ing. Matěj Rejnoch

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky ID16 OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017

.....

Craneballs s.r.o.
Antala Staška 1859/34
140 00 PRAHA 4
IČ: 03031179 DIČ: CZ03031179

Rád bych poděkoval vedení firmy Craneballs s.r.o, za umožnění vykonání odborné praxe a získání podkladů pro mou bakalářskou práci. Také bych rád poděkoval svému vedoucímu práce, panu doc. Mgr. Jiřímu Dvorskému, Ph.D. za vedení a rady. Dále pak svému konzultantovi, panu Ing. Matěji Rejnochovi a v neposlední řadě také všem, kteří mi pomohli k dokončení této práce.

Abstrakt

Tato práce se zabývá popisem individuální odborné praxe vykonávané ve firmě Craneballs s.r.o [1]. Po krátkém úvodu v kapitole 1 následuje popis firmy, ve které jsem pracoval na kapitole 2.1, přes rozbor technologií využitých při vývoji v kapitole 4. Kapitola 3 ve zkratce popisuje jednotlivá zadání úkolů, na kterých jsem po dobu mého působení ve společnosti pracoval, s vyjádřením jejich časové náročnosti a následným detailním popisem a řešením v kapitole 5. Práce je zakončena závěrečnou kapitolou 6, v níž je doplněno, kterých znalostí jsem při vývoji využil, které mi naopak chyběly a celkové zhodnocení bakalářské praxe.

Klíčová slova: Unity game engine, virtuální realita, HTC Vive, simulace, Craneballs, shader

Abstract

This bachelor thesis deals with the description of the individual professional practice performed in Craneballs s.r.o [1] company. After a brief introduction, in the chapter 1, I follow the description of the company, in which I worked in the chapter 2.1, through the analysis of the technologies used in the development in the chapter 4. The chapter 3 briefly describes the individual assignments I have been working on during my work in the company, expressing their time consuming requirements, and the detailed descriptions and solutions in chapter 5. The work is completed with the final chapter 6, which is supplemented with knowledge I used in the development, which I missed and the overall evaluation of the bachelor's practice.

Key Words: Unity game engine, virtual reality, HTC Vive, simulation, Craneballs, shader

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

1 Úvod **11**

2 Popis odborného zaměření firmy a pracovního zařazení **12**
2.1 Odborné zaměření firmy 12
2.2 Popis pracovního zařazení 12

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti **13**
3.1 Návrh a zobrazení neviditelných objektů při doteku ruky 13
3.2 Přenesení scény do virtuální reality 13
3.3 Úchop předmětu do ruky a chování uchopených předmětů 13
3.4 Vyjádření časové náročnosti 13

4 Využité technologie **14**

5 Řešení jednotlivých úkolů **15**
5.1 Návrh a zobrazování neviditelných objektů 15
5.2 Přenesení scény do virtuální reality 20
5.3 Úchop a chování uchopených předmětů v rukách 22

6 Závěr **25**
6.1 Uplatnění teoretické a praktické znalosti 25
6.2 Scházející znalosti 25
6.3 Dosažené výsledky v průběhu praxe a jejich zhodnocení 25

Literatura **26**

Seznam použitých zkratek a symbolů

2D	–	two dimensions
3D	–	three dimensions
PC	–	personal computer
VR	–	virtual reality
SW	–	software
FPS	–	frames per second
GLSL	–	OpenGL Shading Language
HLSL	–	High Level Shading Language

Seznam obrázků

1 Ukázka objektů zobrazených efektem shaderu 15
2 Ukázka prostoru virtuální reality v Unity 21
3 Ukázka úchopu do ruky a chování uchopených předmětů 23

Seznam výpisů zdrojového kódu

1 Ukázka kódu zápisu struktury vlastností shaderu pro Unity s jeho základním nastavením a deklarací proměnných 17
2 Ukázka kódu pro vertex a surface shader 19
3 Ukázka kódu použití trigger buttonu ovladače a vibrací 22
4 Ukázka kódu chování uchopených dveří do ruky 24

1 Úvod

Při výběru bakalářské práce jsem zvolil jednu z nabízených možností, a to konkrétně absolvování individuální bakalářské praxe ve firmě Craneballs s.r.o.[1]. Učinil jsem tak proto, že jsem již delší dobu chtěl získat nějaké zkušenosti ze skutečného prostředí programátorského vývoje a naučit se novým znalostem od skutečných profesionálů v oboru. Jelikož od útlého mládí tíhnu k vývoji videoher, byla pro mě tato možnost skvělou příležitostí, jak si dávný sen splnit a pracovat na projektu společně s týmem podobně nadšených lidí.

Po dobu mého působení ve firmě jsem pracoval na projektu Blind World, který má v kombinaci výkonného desktopu a headsetu virtuální reality simulovat herním způsobem prostředí člověka handicapovaného slepotou.

Nejprve v kapitole 2.1 popíšu odborné zaměření firmy Craneballs [1]. Poté v kapitole 2.2 přiblížím mé pracovní zařazení ve firmě. Zmíním několik z řad úkolů, kterými jsem byl vedením pověřen v kapitole 3 a tyto úkoly pak následně detailně rozeberu v kapitole 5. V závěru zhodnotím množství získaných zkušeností a také porovnám, které předměty ve škole byly pro mě při plnění úkolů klíčové.

2 Popis odborného zaměření firmy a pracovního zařazení

2.1 Odborné zaměření firmy

Firma Craneballs[1] se zabývá vývojem her pro mobilní, ale i desktopová zařízení a v mém případě dokonce i pro desktopová zařízení virtuální reality. Studio vzniklo již v roce 2008, kdy se však soustředilo především na vývoj pro platformy s operačním systémem iOS [15] od společnosti Apple [14]. Mezi jedny z prvních her se řadí Blimp : The Flying Adventures, která se kromě platformy iOS [15] později dočkala i rozšíření na herní konzole PlayStation Portable a PlayStation 3 [11]. Dále pak 33rd Division, Monorace, Fish Heroes, Splash Cars, Ninja Madness nebo SuperRope. V roce 2011 studio vydalo hru Overkill, jež brzy sklidila velmi pozitivní ohlasy jak ze strany hráčů, tak i ze strany kritiky, a zaujala jednu z prvních příček na obchodním portálu společnosti Apple [14] na App Store. Tato úspěšná hra se dočkala i několika pokračování v podobě Overkill 2, úspěšného Overkill 3 anebo Overkill Mafia. V současné době firma vyvíjí projekt Planet Nomads určený pro desktopová zařízení a několik dalších menších projektů určených spíše pro mobilní zařízení.

2.2 Popis pracovního zařazení

Do firmy byl společně se mnou přijat ještě jeden kolega, který byl v té době součástí vývoje SW pro jinou firmu vyvíjející aplikaci pro nemocnice. Z důvodu nedostatku času a obav z nesplnění podmínek se nakonec rozhodl se mnou nespolupracovat, a proto jsem projekt v úvodu vyvíjel sám a učil se, jak v Unity game engine [2] pracovat. Brzy však do mého týmu přibyli dva kolegové programátoři, kteří mi ochotně pomáhali a vedli mě cestou, kterou by se měl projekt vyvíjet. Celý tým se pravidelně scházel a členové konzultovali, zdali se projekt vyvíjí správným směrem, či je potřeba provést nějaké konkrétní změny.

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti

3.1 Návrh a zobrazení neviditelných objektů při doteku ruky

Jedním ze zadaných úkolů bylo navrhnout a implementovat řešení, které by po doteku ruky s neviditelným předmětem ve scéně tento předmět částečně zobrazilo a odkrylo hráči část scény. Tento zobrazený předmět by byl viditelný pouze po dobu kolize ruky s tímto předmětem.

3.2 Přenesení scény do virtuální reality

Dalším úkolem bylo přenesení scény z obrazovky počítače na zařízení HTC Vive [8].

3.3 Úchop předmětu do ruky a chování uchopených předmětů

Další úkol, který popíšu bylo implementovat chování dveří kuchyňské linky, šuplíků, sporáku a dalších zařízení ovlivněných úchopem ruky.

3.4 Vyjádření časové náročnosti

Seznam úkolů	Časová náročnost (hodiny)
Návrh a zobrazování neviditelných objektů	150
Přenesení scény do virtuální reality	40
Úchop a chování uchopených předmětů v rukách	120

4 Využité technologie

Studio již delší dobu vyvíjí SW v Unity game enginu [2], který značně usnadňuje práci, uchovává projekt mnohem přehlednější a přidává možnost měnit hodnoty proměnných nebo parametry v reálném čase. Mezi další výhody tohoto enginu patří zcela jistě i multiplatformní vývoj aplikací, který umožňuje vytvářet finální projekty pro řadu zařízení. Mezi ně se řadí například PC, PlayStation [11] a jeho herní konzole, Xbox[12] a jeho herní konzole, Tizen [13], Android [16], iOS [15], WebGL [17] pro webové prohlížeče, nVidia SHIELD [19] a spoustu dalších, které používají různé zobrazovací metody a knihovny. Toho jsme využili i při vývoji aplikace pro PC a pro VR. V té době byla aktuální verze Unity 3D 5.4.0 [2], která byla ještě částečnou betaverzí, a proto velká část studia zůstávala u otestované verze 5.3.5, stejně jako my. Brzy však nastaly problémy při předávání hodnot do shaderu, které již efektivně řešila verze 5.4.0, a to byl nakonec důvod, proč jsme museli zvolit novou neotestovanou verzi. Game engine Unity [2] je poměrně mladý a stále se rozšiřující SW, který má však oproti konkurenčním enginům v podobě Unreal Engine [3] nebo Cry Engine[4] obrovskou komunitu a studio jej zvolilo již dříve z finančních důvodů, příjemného uživatelského prostředí a neustálého vyvíjení SW.

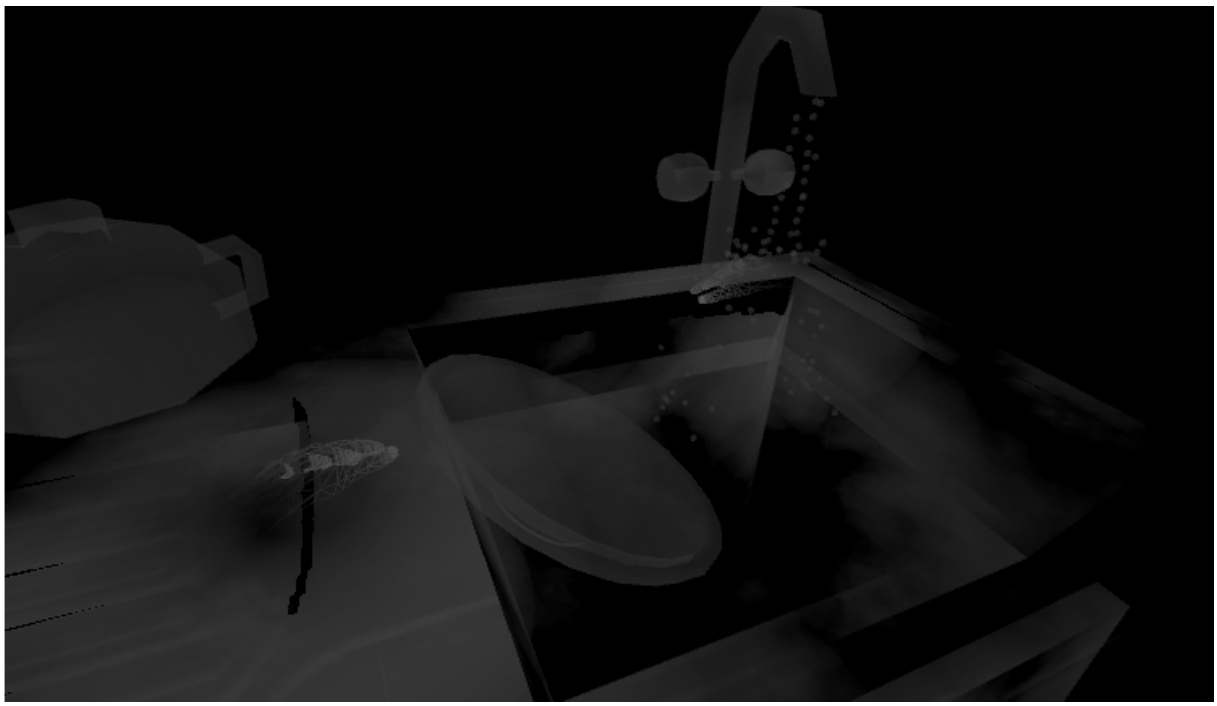
Jako programovací jazyk při vytváření scriptů Unity [2] nabízí použít C# nebo Javascript. Studio vyvíjí SW spíše v C#, proto jsme se i my rozhodli zůstat u C# a aktualizovat a uchovávat kód ve verzích pomocí Bitbucket [5] a SourceTree [6].

5 Řešení jednotlivých úkolů

5.1 Návrh a zobrazování neviditelných objektů

Již v úvodu vývoje aplikace pro virtuální realitu jsme věděli, že by výsledná scéna měla být v kontrastu bílé a černé barvy, abychom se co nejvíce přiblížili simulaci života slepé osoby. Jediné, co by mělo být ve scéně viditelné pro hráče, by měly být modely rukou a v případě doteku ruky s předmětem ve scéně část tohoto předmětu hráči zobrazit.

S týmem jsme přemýšleli a navrhovali řešení tohoto problému a dospěli ke dvěma způsobům, jak scénu zobrazit. První by znamenala použít systém částic (particle system) Unity engine [2], zaměřit se na vertexy objektů a na tyto vertexy umístit texturu. Tuto texturu ještě dále přes bloom efekt světla a kamery přesvítit a vytvořit tak výsledný efekt. Druhá možnost byla vytvořit shader, což je program pro grafickou kartu, který by uměl v místě doteku vytvořit vlnu přes objekt, část předmětu zobrazit a poté zase zmizet, aby se objekt stal opět neviditelným. Vše by mělo být podpořeno ještě vibracemi do ovladačů systému HTC Vive [8] a tím dodat hráči větší pocit z doteku v neviditelném světě. Obě možnosti jsou ve výsledku velmi podobné, jenom efekt zobrazení je jiný a my jsme se museli rozhodnout, která alternativa bude ve finálním produktu lépe vypadat. Proto se tým rozdělil na dvě části, v jedné kolegové pracovali na částicovém efektu a já dostal za úkol zkompletovat shader.



Obrázek 1: Ukázka objektů zobrazených efektem shaderu

Ze školy jsem již měl nějaké zkušenosti s prací na vertex a fragment shaderu s použitím jazyka GLSL knihovny OpenGL (Open Graphics Library) [9]. Unity [2] nabízí možnost vytvářet vlastní

shadery za použití jazyka CG [21] vyvinutého společnosti nVidia [18], což je jakási alternativa mezi jazykem GLSL OpenGL [9] a HLSL DirectX [10]. Jazyk CG [21] společně s Unity [2] vytvoří finální překlad pro systémy užívající DirectX [10] nebo OpenGL [9] a umožňuje vytvářet spoustu subshaderů, které se použijí pro zařízení, na kterém je výsledný projekt spuštěn.

Jako první krok bylo potřeba evidovat místa doteku na objektu. Použil jsem tedy komponenty *Rigidbody* Unity [2] a dodal tak fyzikální chování tělesům. Objekty se poté chovají podle fyzikálních zákonů proto, protože *Rigidbody* používá fyzikální engine PhysX [20]. Poté jsem vytvořil script s eventy *OnCollisionEnter*, *OnCollisionStay* a *OnCollisionExit* spolupracující s fyzikou *Rigidbody* a volající se vždy podle toho, ve kterém stavu se kolize s objektem nachází. Tyto eventy v sobě uchovávají jako parametr kolizní informaci, ze které jsem dále zpracoval třísložkový vektor bodu kolize ve *WorldSpace* a následně tento výsledek předal řídícímu objektu ve scéně. Tento objekt je v hiearchii objektů nad objekty ve scéně a zprostředkovává komunikaci se shaderem. Veškeré změny a požadavky jdou právě přes tento řídící objekt, aby se zajistila určitá optimalizace a nenáročnost výpočtu.

```
Shader "Custom/SpreadWaveShader" {
   Properties {
      _ObjectAlpha("Object aplha", RANGE(0.0, 1.0)) = 0.0
      _Color("Main color", COLOR) = (1.0, 1.0, 1.0, 1.0)
      _DiffuseTex("Diffuse texture", 2D) = "" {}
      _CutOff("Cut off", RANGE(0.0, 1.0)) = 0.1
      _BumpTex("Bump texture", 2D) = "" {}
      _BumpPower("Bump power", FLOAT) = 1.0
      _NoiseTex("Noise texture", 2D) = "" {}
      _NoisePower("Noise power", FLOAT) = 1.0
      _SpreadRange("Spread range", FLOAT) = 0.5
      _SpreadSpeed("Spread speed", FLOAT) = 0.5
   }
   SubShader {
      Tags{"RenderType"="Transparent" "Queue"="Transparent"}

      CGPROGRAM
      #pragma surface surf Lambert vertex:vert alpha

      fixed _ObjectAlpha;
      fixed4 _Color;
      half _BumpPower;
      half _NoisePower;
      sampler2D _DiffuseTex;
      sampler2D _BumpTex;
      sampler2D _NoiseTex;
      half _SpreadRange;
      half _SpreadSpeed;
      half _ActualMaxWaveSpread[10];
      half _AlphaAttenuation[10];
      float _Timers[10];
      half4 _ContactPoints[10];
      float _ActualPointsDistances[10];
      fixed _MaxAlpha = 0.0;
      fixed _CutOff;
```

Výpis 1: Ukázka kódu zápisu struktury vlastností shaderu pro Unity s jeho základním nastavením a deklarací proměnných

Když už jsem znal body kolize v prostoru, bylo potřeba vytvořit samotný shader. Tam pak evidovat strukturu vlastností mezi něž patří barva, alfa kanál, difúzní, normálová textura, textura šumu (noise) s jejími různými nastaveními o síle šumu a celkové nastavení výsledné vlny přes objekt, jako je velikost či rychlost průběhu vlny. Tyto vlastnosti můžeme libovolně měnit před spuštěním aplikace, či v případě běhu programu v debuggovém režimu také realtimeově. V kódu pro vertex shader bylo nutné zpracovat údaje o poloze vertexů objektu a předat je dále do surface shaderu. V surface shaderu jsem nejprve zpracoval *uv* souřadnice objektu, což jsou souřadnice, podle kterých se mapuje textura na objekt, a poté texturu samotnou. Takto surface shader zpracuje textury, které chceme na objekt přidat, a namapuje je na objekt podle těchto kritérií. Difúzní textura je klasická vzhledová textura, normálová dodává imitaci nerovností na objektu a textura šumu najde využití až po vytvořené vlně přes objekt. Po zpracování textur bylo nezbytné vymyslet způsob jak vytvářet vlny. Ty jsem se rozhodl vytvořit tak, že mezi místem kolizního bodu získaného od řídícího objektu, budu zvětšovat pomyslný prostor zvětšující se v čase a interpolovat v tomto prostoru velikost útlumu alfa složky barvy. V nejméně vzdáleném bodu od bodu kolize je alfa složka nejmenší tedy 0.0 a naopak v nejvzdálenějším bodu z pomyslného prostoru je 1.0. Každému zpracovanému bodu jsem pak přiřadil jeho vypočtenou hodnotu alfa složky a objekt se pak stal částečně viditelným. Aby se vlna nezvětšila až do nekonečna, ovlivnil jsem alfa kanál bodů objektu ještě časovou hodnotou a dodal tak efekt mizení vlny v čase. V tento okamžik našla využití šumová textura, která kromě času vlnu také ovlivnila nepravidelností. Z textury šumu jsem zjistil hodnotu barevné složky bodu na objektu, která je od 0.0 do 1.0, a tu pak odečetl od vypočtené velikosti afla útlumu.

```
struct Input {
   float4 localPos;
   float2 uv_DiffuseTex;
   float2 uv_BumpTex;
   float2 uv_NoiseTex;
};
void vert(inout appdata_full v, out Input o){
   UNITY_INITIALIZE_OUTPUT(Input, o);
   o.localPos = v.vertex;
}
void surf (Input IN, inout SurfaceOutput o) {
   float3 _DiffuseTexture = tex2D(_DiffuseTex, IN.uv_DiffuseTex);
   float3 _NormalTexture = UnpackNormal(tex2D(_BumpTex, IN.uv_BumpTex));
   float3 _NoiseTexture = tex2D(_NoiseTex, IN.uv_NoiseTex);

   if (_DiffuseTexture.r < _CutOff) discard;

   _NormalTexture.r *= _BumpPower;
   _NormalTexture.g *= _BumpPower;
   _NormalTexture.b *= _BumpPower;

   _MaxAlpha = max(wave(0, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(1, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(2, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(3, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(4, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(5, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(6, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(7, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(8, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);
   _MaxAlpha = max(wave(9, IN.localPos, _NoiseTexture, _MaxAlpha), _MaxAlpha);

   o.Alpha = _MaxAlpha + _ObjectAlpha;
   o.Albedo = _Color * _DiffuseTexture;
   o.Normal = normalize(_NormalTexture);
}
```

Výpis 2: Ukázka kódu pro vertex a surface shader

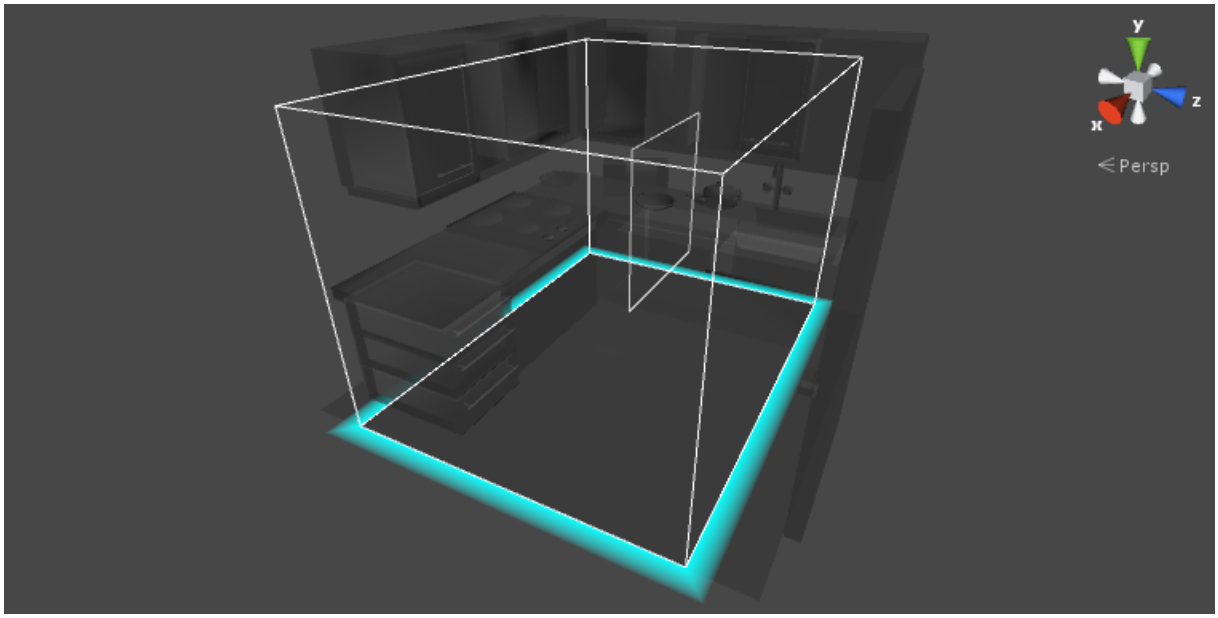
Zpracováváme pole deseti kolizních bodů, tudíž pole deseti vln, u kterých, když dojde k prolnutí, intersekci, tak se jejich hodnota alfa kanálu v daném místě sčítá. Alokaci na právě deset vln jsme se rozhodli zvolit pro rostoucí náročnost výpočtu s přibývajícími vlnami. Pokud dojde k vytvoření jedenácté vlny na objektu, je vlna, která byla vytvořena jako první, zničena a nahrazena novou vlnou a tak dále. Tato situace se v praxi spíše nestává, ale pokud k ní dojde, je to jistá část ošetření, které si uživatel pro velké množství vln na objektu ve většině případů nevšimne. Původně jsem shader rozšířil i o dynamicky počítaný šum pomocí algoritmu *Perlinnoise*, ale tato možnost musela být odstraněna a nahrazena způsobem s šumovou texturou z důvodu optimalizace výpočtu a zvýšení FPS. Těchto optimalizací bylo více. Mezi ně bych zařadil i odstranění for cyklů nebo odstranění if a *else*, které výpočet značně zpomalovaly, a omezení se pouze na matematické výpočty.

Každý předmět, který má být neviditelný a má být ovlivněn efektem vlny při doteku, musí mít materiál, který pro zobrazování užívá náš shader. Zde se nabízela možnost rozdělit objekty na ty, které budou používat *sharedMaterial* (sdílený materiál) a obyčejný *material*, aby se tak zajistila optimalizace aplikace, snížení počtu vytvořených instancí shaderu, a tím snížení počtu průchodů shaderu při vykreslování finální scény. Řídící objekt pak do instancí shaderu rozešle kolizní bod v prostoru, kterých při vyšším počtu doteků může být více.

5.2 Přenesení scény do virtuální reality

Projekt Blind World je primárně určen pro zařízení virtuální reality HTC Vive [8], proto bylo nezbytně nutné nalézt řešení, jak scénu přemístit z obrazovky počítače právě na toto zařízení, dostat uživatele do prostoru vymodelované scény, nastavit chování tlačítek ovladačů a přidat vibrace, aby byl pocit z virtuální reality co možná nejvěrohodnější.

Jak jsem již zmiňoval, Unity [2] má velkou komunitu uživatelů, kterou kromě Unity fóra [2] a dalších jiných komunikačních prostředků sdružuje také Asset store. V něm můžeme najít množství pluginů, modelů, animací, skriptů, částicových efektů a mnoho dalších komponent třetích stran, které lze pořídit zcela zdarma nebo za určitou cenu. Jedním z těchto pluginů, které nabízí společnost Steam je plugin SteamVR [7] pro Unity Engine [2], což je jakýsi ovladač, který zprostředkovává komunikaci mezi Unity [2] a zařízením HTC Vive [8]. Plugin obsahuje demo scénu, skripty, objekty a mnoho dalšího a na nás už je, jak s těmito komponentami naložíme.



Obrázek 2: Ukázka prostoru virtuální reality v Unity

Pro náš projekt byly podstatné pouze určité skripty a objekt *CameraRig*, který nastavoval základní scénu, přenášel informace o poloze ovladače a headsetu v prostoru a šla skrze něj veškerá komunikace mezi HTC Vive [8] a Unity [2]. Do hierarchie tohoto objektu jsem vhodně umístil modely rukou, aby byly zobrazovány přesně na místě ovladače v prostoru snímaného kamerou. Rozhodli jsme se mít prozatím čtyři modely rukou, dva pro levou a dva pro pravou ruku, než se někdy později začneme zabývat animací ruky a plynulostí pohybu. Každá ruka má tedy dva modely, dva stavy, kdy je ruka rozevřená nebo zavřená. Tyto modely bylo potřeba měnit v závislosti na tom, zdali uživatel drží tlačítko pro úchop nebo ne. Kontrolu nad tlačítky ovladače zprostředkovává třída *Steam_Controller.Device*, která pomocí metody *GetPress* vrací, zdali bylo tlačítko skutečně stlačeno, a podle toho stačí pouze vizuálně přeměnit jeden model ruky za druhý. Tato třída také spravuje odezvu vibrací v ovladačích, a to pomocí metody *TriggerHapticPulse*, která jako vstupní parametr přijímá délku pulsu v milisekundách a tím následovně mění velikost pulsu v ovladači.

```
private bool triggerButtonPressed = false;

[HideInInspector]
   public SteamVR_Controller.Device controller { get { return
       SteamVR_Controller.Input((int)trackedObj.index); } }
[HideInInspector]
   public Valve.VR.EVRButtonId triggerButton = Valve.VR.EVRButtonId.
       k_EButton_SteamVR_Trigger;

void Update() {
   if (triggerButtonPressed != controller.GetPress(triggerButton))
      triggerButtonPressed = controller.GetPress(triggerButton);
   if (gameManagerScript.handCollidingObject.Count > 0) {
      ushort maxObjectReaction = 0;
      for (int i = 0; i < gameManagerScript.handCollidingObjects.Count; i++) {
         ushort objectReaction = gameManagerScript.handCollidingObjects[i].
             GetComponent<GameObjectInformations>().objectReaction;

         if (objectReaction > maxObjectReaction)
            maxObjectReaction = objectReaction;
      }
   controller.TriggerHapticPulse(maxObjectReaction);
   }
}
```

Výpis 3: Ukázka kódu použití trigger buttonu ovladače a vibrací

5.3 Úchop a chování uchopených předmětů v rukách

Nejprve bylo potřeba předmětům ve scéně přiřadit jejich chování po úchopu ruky. Proto jsem se rozhodl u předmětů evidovat, zdali lze předmět pouze uchopit do ruky a někam přenést, tahat za něj, tahat a zároveň s ním otáčet, jenom s ním otáčet nebo předmět vůbec neuchopit. Tyto situace zahrnují chování hrnku v ruce, tahání za šuplík, otáčení dveří nebo dvířek kuchyňské linky, chování spínacího knoflíku sporáku či vodovodního kohoutku nebo pevně ke zdi připevněného kuchyňského nábytku.

Obrázek 3: Ukázka úchopu do ruky a chování uchopených předmětů

Na každý z těchto objektů s výjimkou těch, se kterými nelze pohybovat, bylo potřeba umístit bod úchopu. Ten kromě toho, že napomáhal s úchopem samotným, pomáhal také při některých výpočtech rotací objektu. Jelikož Unity [2] pracuje se systémem hierarchie objektů, stačilo předměty, které lze pouze přenášet, umístit pod objekt ruky v hierarchii objektů, posunout a otočit objekt správných směrem ve směru ruky, a tak docílit, že se bude předmět vždy pohybovat s pohybem ruky. U předmětů, za které lze tahat, jsem přidal evidenci osy, po které se předmět pohybuje, a maximální a minimální velikost tahu za předmět. Předmět, za který se tahá a zároveň se otáčí, eviduje osu otáčení, osu tahání, maximální a minimální rotaci, původní pozici bodu úchopu při spuštění scény a aktuální pozici bodu úchopu v ruce. Původně jsem chtěl systém tahání a zároveň otáčení vyřešit použitím komponenty *Hinge joint*, což je způsob, kdy se na objekt, se kterým chceme rotavat, umístí bod, nastaví se osa otáčení a o tuto osu objekt rotuje. Je to rychlé a funkční řešení, které navíc vypadá velmi pěkně, protože *Hinge joint* na sebe váže i komponentu *Rigidbody* a ta dodává tělesům chování podle fyzikálních zákonů. Brzy jsem narazil na podstatný problém, kdy nastávala kolize mezi rukou s předmětem, jenž se začal jakoby klepat v ruce. Musel jsem se tedy nějak vyhnout řešení s použitím fyziky a omezit se pouze na skripty a tento posun spočítat.

```
case GameObjectInformations.gameObjectBehavior.PULLROTATE:
   Vector2 vectorU, vectorV;

   thingToKeepGrab.transform.position = changeHandsScript.holdingHandCenter.
       transform.position;

   switch (objectInformationsScript.pullAxe) {
      case 'x':
      switch (objectInformationsScript.rotationAxe) {
         case 'y':
            vectorU = new Vector2(thingToKeep.transform.position.x, thingToKeep.
                transform.position.z) - new Vector2(objectInformationsScript.
                grabPointDefaultPosition.x, objectInformationsScript.
                grabPointDefaultPosition.z);

            vectorV = new Vector2(thingToKeep.transform.position.x, thingToKeep.
                transform.position.z) - new Vector2(thingToKeepGrab.transform.
                position.x, thingToKeepGrab.transform.position.z);

            thingToKeep.transform.localEulerAngles = new Vector3(
            thingToKeep.transform.localEulerAngles.x,
            Vector2.Angle(vectorU, vectorV),
            thingToKeep.transform.localEulerAngles.z);
      break;
```

Výpis 4: Ukázka kódu chování uchopených dveří do ruky

Ze středu otáčení předmětu bylo nutné vést pomyslné vektory u a v. Vektor u k původnímu bodu úchopu předmětu a vektor v k pozici uchopující ruky. Mezi oběma vektory se již snadno spočítal úhel, v našem případě ve stupních, vypočtený úhel je tedy rotací předmětu. Důležité také bylo si tyto vektory u a v vytvářit v závislosti, o kterou osu předmět rotuje a po které ose se s předmětem manipuluje, protože s předmětem pohybujeme v rovině 2D prostoru, nikoliv 3D. Řešení v 3D rovině by se nabízelo, pokud bychom rotovali v kombinaci os například x a y nebo jiných, pro nás však rotace o základní osy byly více než dostačující. Při pouhé rotaci objektu jsem spočítal rozdíl mezi rotací ruky v aktuálním stavu a rotací v předchozím stavu a o tento rozdíl se změnila rotace objektu. Při práci na chování uchopených předmětů bylo potřeba rozsáhlých změn v počítání bodu kolize ruky s předměty. Zjišťovat tento bod nestačilo pouze z výpočtů fyziky Unity, ale musela se přidat také metoda paprsku *BoxCast* a *RayCast*, která pracovala i s předměty statickými, neovlivněnými fyzikálními zákony.

6 Závěr

6.1 Uplatnění teoretické a praktické znalosti

V průběhu absolvování individuální bakalářské praxe jsem využil hned několika znalostí získaných během studia. Základní kámen programování v Unity Engine [2] byl v našem případě jazyk C# (PJ II), avšak v průběhu hledání vhodného řešení pro problémy, které se vyskytly, jsme několikrát museli zavítat do odvětví Unity fóra s Javascriptem (TAMZ I). Užitečná byla zcela jistě i znalost a použití různých algoritmů v praxi (ALG I, ALG II), práce s grafikou a shadery (ZPG) a znalost úpravy modelů (MGA) při vytváření finální scény.

6.2 Scházející znalosti

Až ve firmě Craneballs [1] jsem se setkal s vývojem v herním enginu, se kterým jsem do té doby neměl vůbec žádné zkušenosti. Kromě rozšíření znalostí o programovacím jazyku C# jsem se do té doby nesetkal s zařízením HTC Vive [8] z vývojové stránky hlediska. Jazyk CG [21] grafických shaderů byl pro mě také úplnou novinkou.

6.3 Dosažené výsledky v průběhu praxe a jejich zhodnocení

Absolvování individuální bakalářské praxe se pro mne stalo velmi obohacující zkušeností jak z hlediska programového vývoje, tak z hlediska práce v týmu, fungování ve firmě a práce na reálných komerčních projektech. V současné době se stále více a více setkáváme se zařízeními virtuální reality, a proto zkušenost s vývojem právě na tato zařízení je více než přínosná a obohacující. Pro společnost Craneballs [1] dále pracuji, jsem velmi vděčný za příležitosti, kterých se mi dostalo, a celou praxi hodnotím zcela pozitivně.

Literatura

[1] Craneballs [online]. © 2012 [cit. 2017-04-01].
Dostupné z: http://www.craneballs.com/

[2] Unity – Game engine [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://unity3d.com/

[3] Unreal Engine [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.unrealengine.com/what-is-unreal-engine-4

[4] Cry Engine [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.cryengine.com/

[5] Bitbucket [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://bitbucket.org/

[6] Atlassian SourceTree [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.sourcetreeapp.com/

[7] SteamVR [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://steamcommunity.com/steamvr

[8] HTC Vive [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.vive.com/eu/

[9] OpenGL [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.opengl.org/

[10] DirectX [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=17431

[11] Playstation [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.playstation.com/cs-cz/

[12] Xbox [online]. © 2017 [cit. 2017-04-01].
Dostupné z: http://www.xbox.com/cs-cz

[13] Tizen [online]. © 2012 [cit. 2017-04-01].
Dostupné z: https://www.tizen.org/

[14] Apple [online]. © 2017 [cit. 2017-04-01].
Dostupné z: http://www.apple.com/cz/

[15] iOS [online]. © 2017 [cit. 2017-04-01].
Dostupné z: http://www.apple.com/cz/ios/ios-10/

[16] Android [online]. © 2014 [cit. 2017-04-01].
Dostupné z: https://www.android.com/

[17] WebGL [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.khronos.org/webgl/

[18] nVidia [online]. © 2017 [cit. 2017-04-01].
Dostupné z: http://www.nvidia.com/page/home.html

[19] nVidia SHIELD [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://www.nvidia.com/en-us/shield/

[20] nVidia PhysX [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://developer.nvidia.com/physx-sdk

[21] nVidia CG Toolkit [online]. © 2017 [cit. 2017-04-01].
Dostupné z: https://developer.nvidia.com/cg-toolkit